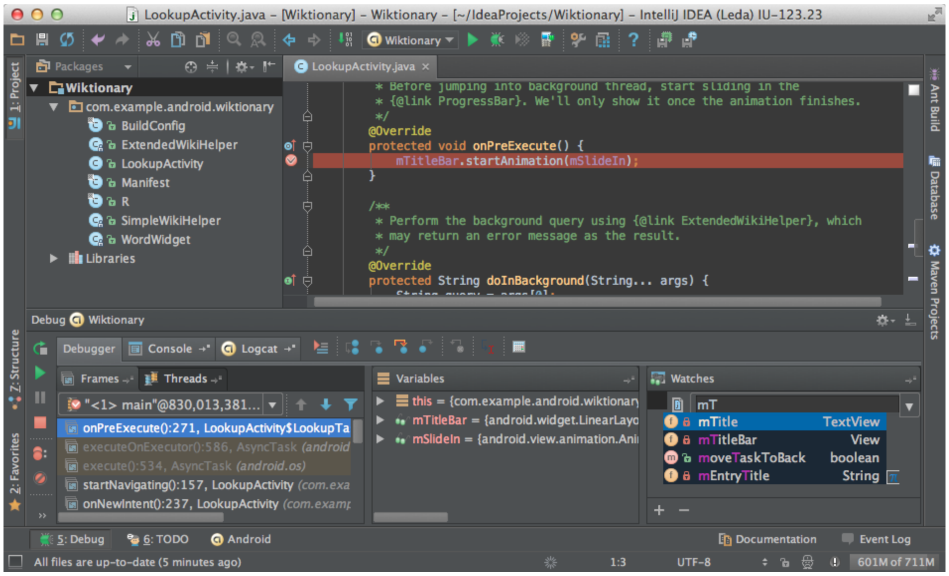
FootPrint - The Data Tracker

When debugging in Java it is difficult to know the state of data structures during runtime in a quick and responsive manner. In modern IDE's during debugging, the developer can view important information such as the current state of variables, and the stack trace of the current execution (Fig 1), but it is impossible to see information about what the variable states were at previous points in the execution. Without plugins, developers often resort to extensive use of debug print statements, a whole problem in itself. There do exist plugins for various popular IDEs that attempt to solve this problem, but fall short in their complexity and bloat, such as JIVE for Eclipse (Fig 2) and Flow for Intellij. What's more, both JIVE and Flow are designed to better display the graphs in the data structures, which is different from FootPrint. FootPrint focuses more on recording the state of all kinds of data structures, variables and functions called. These features would better display the changes in the flow.



LookupActivity.java – [Wiktionary] – Wiktionary – [~/IdeaProjects/Wiktionary] – IntelliJ IDEA (Leda) IU–123.23

```
 * Before jumping into background thread, start sliding in the
 * {@link ProgressBar}. We'll only show it once the animation finishes.
 */
@Override
protected void onPreExecute() {
    mTitleBar.startAnimation(mSlideIn);
}

/**
 * Perform the background query using {@link ExtendedWikiHelper}, which
 * may return an error message as the result.
 */
@Override
protected String doInBackground(String... args) {
```

"<1> main"@830,013,381...

onPreExecute():271, LookupActivity$LookupTa
executeOnExecutor():586, AsyncTask *(android*
execute():534, AsyncTask *(android.os)*
startNavigating():157, LookupActivity *(com.exa*
onNewIntent():237, LookupActivity *(com.examp*

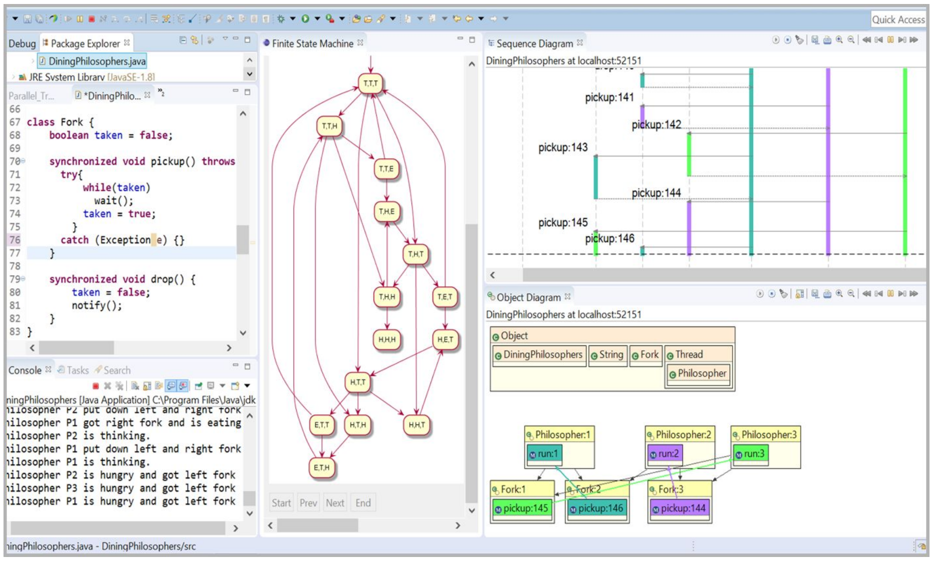
this = {com.example.android.wiktionary
mTitleBar = {android.widget.LinearLayo
mSlideIn = {android.view.animation.Ani

mT

mTitle	TextView
mTitleBar	View
moveTaskToBack	boolean
mEntryTitle	String

All files are up-to-date (5 minutes ago)

Fig 1. Intellij Debugger



```
class Fork {
    boolean taken = false;

    synchronized void pickup() throws
      try{
          while(taken)
            wait();
          taken = true;
        }
      catch (Exception e) {}
    }

    synchronized void drop() {
        taken = false;
        notify();
    }
}
```

```
ningPhilosophers [Java Application] C:\Program Files\Java\jdk
hilosopher P2 put down left and right fork
hilosopher P1 got right fork and is eating
hilosopher P2 is thinking.
hilosopher P1 put down left and right fork
hilosopher P1 is thinking.
hilosopher P2 is hungry and got left fork
hilosopher P3 is hungry and got left fork
hilosopher P1 is hungry and got left fork
```

ningPhilosophers.java - DiningPhilosophers/src

Fig 2. JIVE for Eclipse

Our solution, FootPrint**,** aims to solve this problem by providing an Intellij plugin that allows developers to access and explore what happens during debugging, including dynamic changes in data structures, variables, and functions called. FootPrint will only record this information when debugging is executed and provides it to the developer in their IDE. FootPrint will allow the developer the ability to save past histories persistently to compare to other histories. Further, developers can search in histories for notable events or program states. We envision that FootPrint will greatly aid developers of all skill levels in fixing bugs and finding performance bottlenecks.

One major limitation of FootPrint is the way in which it will need to convey the states of complex data structures. Because the information is text-based, FootPrint will therefore not be a good tool for understanding the layout or design of data structures, rather it is meant to track the changes within them. Further, it will be difficult to display changes in large data structures concisely and in an easily understandable manner, especially in graphs.

Our biggest potential problem is feature creep during development. There are many potentially useful features that could be added, such as tracking of different aspects of a program's environment or tracking changes in the code itself. However, these features are out of the scope of the current project proposal, and including them would certainly delay the project. We will mitigate this issue by clearly defining the requirements for the project and adhering to them.

Time Spent: about 5 hours